RELIABILITY & AVAILABILITY IN THE CLOUD

A TWILIO PERSPECTIVE

September 2012
To the leaders and engineers at Twilio, the cloud represents the promise of reliable, scalable infrastructure at a price that directly reflects a customer’s usage. By aggregating the software requirements of hundreds of thousands of customers, the cloud offers greater reliability than what individual companies can afford to implement independently.

With that philosophy in mind, Twilio has built its cloud infrastructure from the ground up with an emphasis on high availability at an affordable price. We’re proud of our track record and make it publicly available through status.twilio.com (for real-time status) and through an independent third party that monitors the historical uptime of our API.

Twilio’s service is “pay as you go,” which means we only collect revenue when Twilio is successfully processing our customers’ communications. This approach governs how we think about the architecture and operation of our service. Here are some of the design principles and operational practices that we employ:

**CONNECTIONS WITH MULTIPLE TIER 1 CARRIERS**

The worldwide carrier network is complex and constantly changing. To accommodate this dynamic environment, Twilio maintains redundant inbound and outbound connectivity with dozens of network carriers around the world. Our real-time systems dynamically route each call or message via the carrier with the best connectivity at any point in time, responding automatically to carrier availability and reliability.

**MULTIPLE FAULT-INDEPENDENT DATA CENTERS**

Twilio’s infrastructure is hosted in multiple data centers, utilizing state-of-the-art practices for fault tolerance at each level of system infrastructure, including power, cooling and backbone connectivity. We maintain instances of our systems across all of these data centers at all times, all transparently to our customers. We maintain this software distribution across data centers because individual hosts and even entire data centers can experience issues from time to time. If such events occur, Twilio’s software is architected to detect and automatically route around these issues in real time to ensure a consistent customer experience.

**DISTRIBUTED SOFTWARE DESIGN PRINCIPLES**

As companies like Facebook, Google and Amazon have found, distributed systems provide many fault tolerance benefits. However, they require a different set of design principles from traditional monolithic software to operate. Twilio was designed and built as a distributed software system, using the best practices of modern software development to achieve scale and reliability. Twilio codified the following principles to guide all architecture and engineering execution. These represent best practices for distributed system design.
**Service Oriented Architecture**

At the core of our systems is a Service Oriented Architecture, according to which the overarching service Twilio offers to customers is provided by dozens of underlying services. Each of the underlying services operates independently of the other services, and each service provides independent scaling, redundancy and geographic distribution via internal, load-balanced APIs. Each service consumer in the infrastructure has a set of independent local load balancers on the host to avoid single points of failure at the request distribution layer.

**Small stateless services**

We separate Twilio business logic into small stateless services organized into simple homogenous pools of resources. For example, when you make a recording using the `<Record>` verb in TwiML, the work of post-processing the recording to improve the audio quality and upload it to persistent storage is provided by a pool of geographically distributed recording servers. This pool of stateless recording services allows downstream services to retry requests on other instances of the recording service in the event of a service or host failure. In addition, the size of the recording server pool is easily scaled up and down in real time based on load.

**Unit-of-failure is a single host**

No matter how many redundant power supplies, Ethernet ports or CPUs are added, hosts fail for a variety of reasons. Our systems assume that failures are inevitable and are designed to operate in the presence of potential host failures. Each service is spread across a plurality of hosts, across multiple data centers, with fault detection and routing performed in real time by each consuming service. If a host providing a service fails, internal systems automatically route around the failed host until it returns. By building simple services composed of single hosts, orchestrated in a decentralized fashion, we create reliable internal services that can withstand either host or network connectivity failures.

**Share-nothing architecture**

To eliminate points of failure and performance bottlenecks, we avoid shared resources that can often cause cascading failures of many hosts at the same time. For example, we don’t utilize Network Attached Storage for online services because it represents a failure mode that can affect many hosts simultaneously.

**Short timeouts and quick retries**

When failures happen, software quickly identifies those failures and retries requests with alternate hosts. Between internal systems, we implement short timeouts and quick retries at exponential, randomized backoff intervals. Short timeouts allow quick identification of a potential host issue and allow software to quickly retry the request with other hosts in the cluster. Exponentially timed backoffs with random jitter ensure a smooth transition to the other...
hosts in the cluster and help buffer the changing traffic pattern during an automated failover. That may sound like a lot of jargon, but it’s best practice for networked services to ensure reliable recovery when issues occur.

**Idempotent service interfaces**

Due to the quick-retries rule above, we build services that allow requests to be safely retried. The practice is referred to as idempotency. In computer science, the term idempotent is used more comprehensively to describe an operation that will produce the same results if executed once or multiple times. If the API of a dependent service is idempotent, that means it is safe to retry failed requests when a service isn’t sure if a previous request succeeded or failed.

**INFRASTRUCTURE AUTOMATION**

Twilio’s automated testing and deployment systems permit our engineers to deploy new code and database schema changes to our clusters in real time with no manual intervention or production systems. After rigorous automated testing in development and staging, production software updates are deployed by systems that remove hosts from a cluster, apply new software, run a battery of tests and then insert hosts back into the cluster. These “rolling updates” ensure a smooth transition of software versions. Additionally, previous deployments are left intact for a period of 36 hours or more in the event that an online rollback is required.

**Zero Scheduled maintenance**

Twilio doesn’t believe in scheduled maintenance, period. Any updates to our systems are performed without disrupting the online operations of our service. In most companies, the common culprits of “scheduled maintenance” are host or networking changes, as well as large database schema migrations. At Twilio, our systems employ a “rolling deployment” strategy to roll out change in real time on a host-by-host basis. In the event of large database schema changes, we have developed a “database pivot” strategy to apply changes to replicated copies of databases, and then perform an atomic “pivot” from an old host schema to a new host schema, without incurring the downtime traditionally associated with large schema changes.

**Operations Team**

Building self-healing, distributed systems is our goal, and we recognize operations are just as important as infrastructure in crossing the goal line. At all times, Twilio maintains a cross-functional team of operations, engineers and customer support personnel on 24-hour duty to respond to systems and customers in the unlikely event of an issue.

**Transparency**
The relationship between cloud providers and their customers is built on trust and transparency. That's why we publish a real-time status dashboard at status.twilio.com. It is operated on an independent infrastructure, as a mechanism to message real-time service issues either with our infrastructure or any upstream carrier worldwide that may affect your applications. Additionally, we have contracted a trusted third-party, Pingdom, to monitor our infrastructure from around the world every 60 seconds and to publish availability reports for the world to see.

CONCLUSION

In the past, every company was responsible for its own IT infrastructure, and high availability and reliability was only achieved at great expense. At Twilio, we believe that the cloud now makes it possible for every company to enjoy high levels of availability and reliability without large investments in hardware, software and technical expertise.

As described in this paper, Twilio provides an on-demand, distributed, self-healing fault-tolerant cloud communications platform that delivers extreme reliability and availability to our customers. We make this available as a “pay-as-you-go” service so that businesses can have access to the communications services they need, at an affordable price. We invest in infrastructure and redundancy on every level so that you can invest in your business.